



Business Computing
Decision Support & OR Lab

Universität Paderborn

Fachbereich Wirtschaftswissenschaften

Fach Wirtschaftsinformatik

Seminararbeit

Web Services mit PHP

vorgelegt von

Tobias Bräutigam

vorgelegt bei

Prof. Dr. Leena Suhl

Dipl. Wirt. Inform. Alexander Roth

Warstein, den 28.09.2004

Inhaltsverzeichnis:

1	Einleitung	1
2	Grundlagen	1
2.1	Was ist ein Web Service?	1
2.2	XML Grundlagen	2
3	Web Services im Detail	2
3.1	Service orientierte Architektur	2
3.2	Die wichtigsten Technologien	4
3.2.1	UDDI	4
3.2.2	WSDL	5
3.2.3	SOAP	8
3.3	Exkurs: PEAR::SOAP	10
4	Vergleichbare Technologien	11
4.1	XML-RPC	11
4.2	REST	12
4.3	CORBA	12
4.4	Remoting	13
5	Prototypische Umsetzung	13
5.1	Anforderungen	13
5.2	Architektur	14
5.3	Implementierungsdetails	15
5.4	Die Google API	16
5.5	Die Amazon API	17
6	Zusammenfassung und Ausblick	18

1 Einleitung

In den letzten Jahren waren Web Services in aller Munde. Doch obwohl ein regelrechter Hype in diesem Bereich entstanden ist, sind brauchbare Anwendungen auch heute noch rar gesät. In diesem Dokument soll zunächst eine Einführung in Web Services gegeben werden und deren wichtigste Standards vorgestellt werden. Die Implementierung dieser Standards wird anhand des PEAR::SOAP-Toolkits, der Web Service Implementierung in der Programmiersprache PHP, beispielhaft gezeigt. Nach einer kurzen Abgrenzung zu vergleichbaren Technologien wird ein Prototyp vorgestellt, der sowohl als Server, als auch als Client fungiert und einen plattformübergreifenden Zugriff auf Wissensressourcen ermöglicht. Mit Hilfe dieses Prototyps wird eine einheitliche Oberfläche für den Zugriff auf die Web Service Funktionalität der Google- und Amazon-Dienste geschaffen.

2 Grundlagen

2.1 Was ist ein Web Service?

Definition:

“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

([Booth04] Kapitel 1.4)

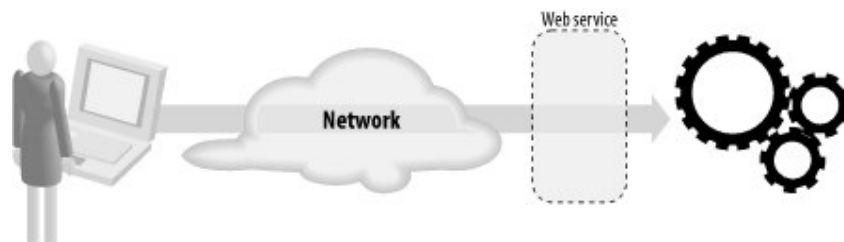


Abbildung 1: Ein Web Service erlaubt den Zugriff auf z.B. die Methoden einer Applikation durch Benutzung der Standard Internet Technologien

Die Idee, die hinter den Web Services steckt ist also, dass definierte Übertragungsstandards dafür sorgen sollen, dass Daten und Funktionalität zwischen Applikationen ausgetauscht werden können, wobei die Kommunikationspartner Maschinen sind, daher spricht man auch von einer Maschine-zu-Maschine Kommunikation.

Diese Idee ist im Grunde nicht neu, da es schon viele verschiedene Ansätze gibt um diese Kommunikation zu verwirklichen, wie z.B. CORBA¹ oder RMI². Das grundsätzlich Neue an dem Konzept der Web Services ist jedoch, dass sie auf etablierten Technologien wie der *Extensible Markup Language* (XML) und dem *Hypertext Transfer Protocol* (HTTP) basieren.

2.2 XML Grundlagen

XML ist aus der *Standart Generalized Markup Language* (SGML) entstanden und durch das *World Wide Web Consortium* (W3C) standardisiert worden. Es bietet dem Autor die Möglichkeit eigene Tags und Attribute zu definieren und somit die Struktur des Dokuments selbst zu bestimmen, wobei die Inhalte komplett von der Formatierung getrennt werden. Während im XML-Dokument nur die Informationen und Strukturen gespeichert werden, werden die Formatierungsanweisungen mit der *eXtensible Stylesheet Language* (XSL) definiert. Die eigentliche Struktur des XML Dokumentes, d.h. die Namen und Formate der Elemente und Attribute und deren Position im Dokument, wird durch das *XML-Schema*, beziehungsweise eine *Document Type Definition* (DTD) beschrieben.

3 Web Services im Detail

3.1 Service orientierte Architektur

Obwohl Web Services auf bekannten „Internet-Technologien“ basieren, unterscheidet sich die dahinter stehende Architektur grundlegend von der klassischen Client/Server Architektur des Internets. Dort greifen die Clients auf bestimmte Daten, bzw. Funktionalität auf einem Server zu und stellen diese im Browser dar. Bei Web Services hingegen findet eine Maschine-zu-Maschine Kommunikation statt. Hier werden Daten bzw. Funktionalität zwischen Anwendungen ausgetauscht.

¹ Siehe Kapitel 4.3 CORBA

² Siehe Kapitel 4.4 Remoting

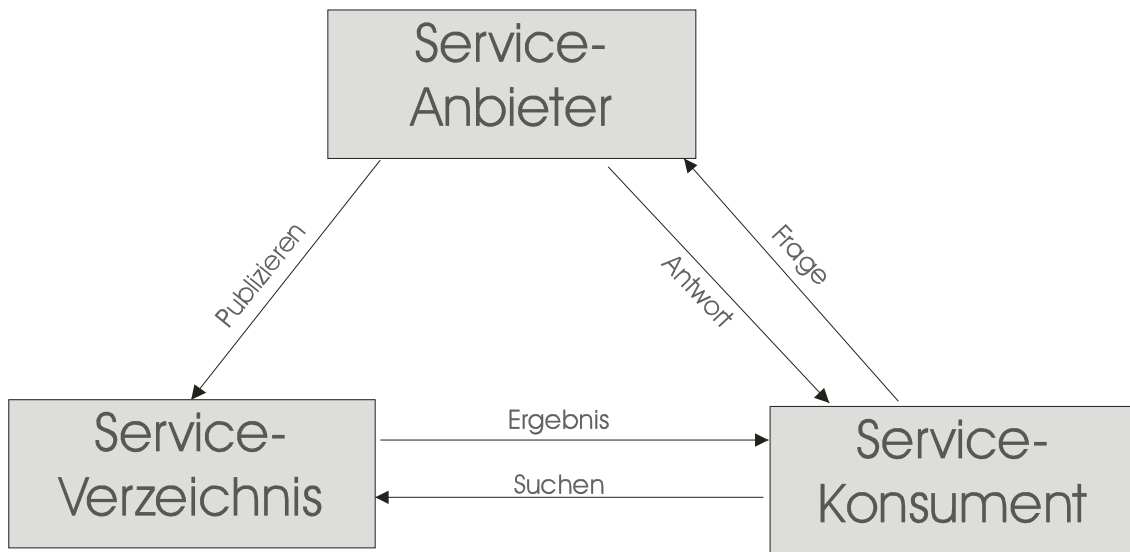


Abbildung 2: Die service-orientierte Architektur

Web Services lassen sich durch die so genannte service-orientierte Architektur (SOA) beschreiben. Diese Architektur basiert auf folgendem Prinzip: Ein Service-Anbieter stellt einen Service zur Verfügung. Um diesen bekannt zu machen veröffentlicht er ihn in einem Service-Verzeichnis. Der Service-Konsument kann nun in diesem Verzeichnis nach dem Service suchen, den er beanspruchen will, und erhält eine URL mit der er ihn beanspruchen kann (siehe Abbildung 2).

3.2 Die wichtigsten Technologien

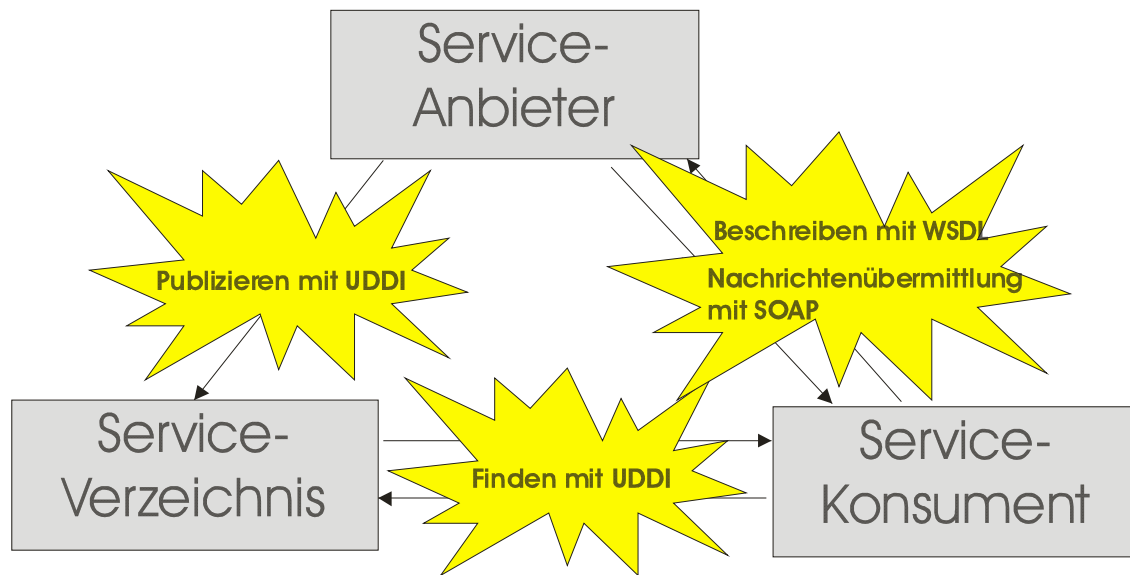


Abbildung 3: Die wichtigsten Web Service – Technologien und deren Einsatzgebiet

3.2.1 UDDI

Vor dem Hintergrund des Konzeptes der service-orientierten Architektur sollen hier die wichtigsten Protokolle die sich in den letzten Jahren durchgesetzt haben vorgestellt werden und zwar in der Reihenfolge, in der sie ein Web Service Konsument benutzen würde.

Zunächst einmal muss der Serviceanbieter seinen Web Service bekannt machen. Zu diesem Zweck hat sich mittlerweile das *Universal Description, Discovery and Integration* Protokoll (UDDI) als De-facto-Standard durchgesetzt. UDDI wird von der *Organisation for the Advancement of Structured Information Standards* (OASIS) gepflegt und wurde von IBM, Microsoft und Ariba ins Leben gerufen.

Wie der Name vermuten lässt ermöglicht das UDDI-Protokoll die Beschreibung, das Auffinden und die Verwendung von Web Services. UDDI ist selbst wiederum eine Web Service-Anwendung und benutzt SOAP³ als Kommunikationsprotokoll.

Ein UDDI-Verzeichnis ist in drei unterschiedliche Bereiche aufgeteilt (vgl. [Newcom02], S. 114f). In den so genannten *White Pages* werden Namen, Beschreibung und Kontaktinformationen zu den Firmen gespeichert. In den *Yellow Pages* werden diese Firmen klassifiziert und

³ Siehe Kapitel 3.2.3 SOAP

kategorisiert, zum Beispiel durch ihre Branchenzugehörigkeit oder geographische Lage, so dass die *Yellow Pages* als Branchenbuch⁴ verstanden werden können.

Abschließend bleiben dann noch die *Green Pages*. Diese beinhalten die technische Beschreibung eines Web Services. Hier wird die Funktionalität eines Web Services beschrieben und ein Link zu dem entsprechenden WSDL-Dokument gesetzt, falls ein solches existiert.

3.2.2 WSDL

Die *Web Services Description Language* (WSDL) basiert auf XML und dient zur Beschreibung von Web Services. Um auf einen Web Service zugreifen zu können sind Detailinformationen über diesen nötig. Ist der Quelltext bekannt stellt dies kein größeres Problem dar, da man alle benötigten Informationen, wie zum Beispiel die Namen der Methoden und deren Parameter ganz einfach im Quelltext nachlesen kann. Was aber ist wenn man auf einen fremden Web Service zugreifen möchte über den man keine weiteren Informationen hat? Um eine Kommunikation auch in diesem Falle zu ermöglichen wurde WSDL eingeführt.

WSDL wird von dem *World Wide Web Consortium* (W3C) verwaltet. Die derzeit aktuellste Version trägt die Nummer 2.0 und hat den Status eines *Working Drafts*, d.h. es handelt sich um einen Arbeitsentwurf, der zu Diskussion freigegeben wurde.

```
<definitions>
  <types/>
  <message/>
  <portType/>
  <binding/>
  <service/>
</definitions>
```

Abbildung 4: Grundstruktur eines WSDL-Dokumentes

Abbildung 4 zeigt den Grundaufbau eines WSDL-Dokumentes. Das Wurzelement `<definitions>` beherbergt fünf weitere Elemente.

In `<types>` werden die in dem Web Service benutzten Datentypen definiert. Diese Definition orientiert sich in der Regel an den in der XML-Schema-Spezifikation verwendeten Typen.

⁴ Vergleichbar mit den Gelben Seiten für Telefonnummern

Das Element `<messages>` beinhaltet Informationen zu den Anfragen und Antworten des Web Services. Mittels `<portType>` werden die Methoden des Web Services beschrieben. In `<binding>` stehen die unterstützten Protokolle wie zum Beispiel SOAP, HTTP GET oder HTTP POST, und `<service>` enthält den *Uniform Resource Locator* (URL)⁵ des Dienstes.

Mittlerweile bieten viele Web Service Toolkits eine automatische WSDL Generierung. Somit muss der Entwickler des Web Services keinerlei Vorarbeit leisten um den Clients ein passendes WSDL-Dokument zur Verfügung zu stellen.

Abbildung 5 zeigt so ein WSDL-Dokument für einen einfachen Web Service, welcher automatisch vom PEAR::SOAP⁶ Toolkit generiert wurde. Dieses Dokument beschreibt einen einfachen Web Service, welcher die Methode `hello` (`<operation name="hello">`) zur Verfügung stellt. Diese Methode erwartet als Parameter einen String und gibt als Ergebnis ebenfalls einen String zurück.

```
<definitions ...>
</types>
<portType name="BezeichnungPort">
  <operation name="hello">
    <input message="tns:helloRequest" />
    <output message="tns:helloResponse" />
  </operation>
</portType>
<binding name="BezeichnungBinding" type="tns:BezeichnungPort">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="hello">
    <soap:operation soapAction="urn:soapservice#service#hello"
  />
  <input>
```

⁵ Eindeutiger Pfad zu einer Ressource, Beispiel: <http://www.google.de>

⁶ Siehe Kapitel 3.3 Exkurs: PEAR::SOAP

```

    <soap:body use="encoded" namespace="urn:soapservice"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    />
  </input>

  <output>

    <soap:body use="encoded" namespace="urn:soapservice"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    />

  </output>
</operation>
</binding>
<service name="BezeichnungService">
  <documentation />
  <port name="BezeichnungPort "
    binding="tns:BezeichnungBinding">
    <soap:address location="http://www.to-bra.de/ws/Service.php"
    />
  </port>
</service>
<message name="helloRequest">
  <part name="format" type="xsd:string" />
</message>
<message name="helloResponse">
  <part name="format" type="xsd:string" />
</message>
</definitions>

```

Abbildung 5: Beispiel für ein WSDL-Dokument, generiert mit PEAR::SOAP, der Quelltext dieses Services ist in Abbildung 8 zu sehen.

3.2.3 SOAP

Das *Simple Object Access Protocol* (SOAP) ist ein Protokoll, welches zur Übermittlung von XML-Nachrichten dient. Im Umfeld der Web Services ist SOAP wohl das bekannteste aller Protokolle. Dies ist vielleicht damit zu begründen, dass SOAP für den wichtigsten Aspekt von Web Services zuständig ist, nämlich wie man die Daten von einem Ort zum anderen übermittelt.

SOAP wird in der aktuellen Version 1.2 wie auch WSDL vom W3C gepflegt und hat den Status einer *Recommendation*. Das ist die höchste Stufe des W3C und gilt damit als Standard. Seit der Version 1.2 wird SOAP nicht mehr als Akronym benutzt, sondern gilt als eigenständiger Begriff (vgl. [Mitra03] Kapitel 6).

Die SOAP-Nachrichten werden in der Regel über HTTP transportiert, es sind allerdings auch andere Übertragungsprotokolle möglich wie zum Beispiel SMTP. Da der Datenverkehr dieser Standard-Internetprotokolle von den meisten Firewalls nicht blockiert wird, können die Web Services ohne weiteres auch hinter einer solchen benutzt werden. Ein weiterer Vorteil, den SOAP mit sich bringt, ist die Plattformunabhängigkeit. Im Idealfall können Anwendungen über SOAP miteinander kommunizieren ohne das Betriebssystem oder verwendete Programmiersprache eine Rolle spielen.

SOAP kennt die zwei Nachrichtenarten (*style*) *RPC* und *document* und die zwei Darstellungsformen (*encodingStyle*) *literal* und *encoded*. Da diese beliebig miteinander kombiniert werden können ergeben sich vier Nachrichten-Modi in SOAP, jedoch sind die beiden Kombinationen *RPC/encoded* und *document/literal* die gebräuchlichsten.

Eine SOAP-Nachricht im *RPC/encoded* Modus beinhaltet einen entfernten Methodenaufruf und enthält den Methodennamen und die entsprechenden Parameter, welche nach bestimmten Vorgaben serialisiert bzw. kodiert wurden (vgl. [Gudgin03] Kap. 3 SOAP Encoding).

Ist die SOAP-Nachricht allerdings im *document/literal* Modus enthält sie ein Fragment eines XML-Dokuments ohne spezielle Kodierung und kann sogar durch ein XML-Schema beschrieben und validiert werden. Es ist ebenfalls möglich einen entfernten Methodenaufruf (RPC) in dieses Dokument einzubetten, wie es zum Beispiel bei den .NET Web Services gemacht wird.

```

1: <?xml version="1.0" encoding="utf-8"?>
2: <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">
3:   <soap:Header>
4:   <soap:Header>
5:   <soap:Body>
6:     <ausgeben xmlns="http://web-services.de/hallowelt">
7:       <text>Welt</text>
8:     </ausgeben>
9:   </soap:Body>
10: </soap:Envelope>

```

Abbildung 6: Beispiel einer SOAP-Nachricht (vgl. [Wenz04] S. 73)

Das SOAP-Format lässt sich analog zu einem Brief beschreiben. Eine SOAP-Nachricht besteht aus drei Teilen: Dem *Envelope*, der die beiden anderen SOAP-Bestandteile enthält, dem *Header*, welcher den Kopfteil der Nachricht enthält und dem *Body*, welcher den eigentlichen Inhalt der Nachricht beinhaltet (siehe Abbildung 6). Da SOAP ein XML-Format ist, beginnt die Nachricht mit dem XML-Header gefolgt von den drei Elementen der SOAP-Nachricht, *Envelope*, *Header* und *Body*. Wie man in diesem Beispiel sieht darf der *Header* auch leer sein und in manchen Fällen wird er sogar ganz weggelassen. Ansonsten können dort beispielsweise Authentifizierungsinformationen untergebracht werden. Der *Body* hingegen muss vorhanden sein. Im Beispiel aus Abbildung 6 wird ein fiktiver Web Service *ausgeben* mit dem Parameter *text* aufgerufen, welcher sich innerhalb des Namespaces⁷ *http://web-service.cc/hallowelt* befindet.

Die Antwort des aufgerufenen Web Services unterscheidet sich von der Struktur her nicht von der Anfrage und wird daher an dieser Stelle nicht dargestellt⁸.

Sollte bei einem SOAP-Aufruf ein Fehler auftreten, bietet SOAP auch eine Fehlerbehandlung an. In diesem Falle wird ein `<Fault>`-Element vom Web Service zurückgegeben (siehe Abbildung 7).

⁷ dient zur Unterscheidung von Elementen und Attributen innerhalb eines XML-Dokumentes

⁸ zu finden in [Wenz04] Seite 75

```
<soap:Fault>
  <faultcode xsi:type="xsd:QName">soap:Client</faultcode>
  <faultstring xsi:type="xsd:string">Ein Fehler ist aufgetre-
ten</faultstring>
</soap:Fault>
```

Abbildung 7: Beispiel eines SOAP-Fehlers

3.3 Exkurs: PEAR::SOAP

Das *PHP Extension and Application Repository* (PEAR) ist eine Sammlung von Bibliotheken und Erweiterungen für die Skriptsprache PHP, vergleichbar mit dem CPAN Projekt für Perl. PEAR::SOAP ist nun die PEAR Variante von SOAP und basiert wie NuSOAP⁹ auch auf SOAPx4¹⁰.

Durch die Vorgaben von PEAR muss ein Web Service komplett objektorientiert implementiert werden.

```
class Service {
    function hello ($string) {
        if ($string == null || trim($string)== "") {
            return new SOAP_Fault("Kein Parameter
angegeben", "0815");
        }
        else {
            return "Hello ".$string."!";
        }
    }
}

require_once ("SOAP/Server.php");
$soap = new SOAP_Server();
```

⁹ Weiterentwicklung von SOAPx4, wird gepflegt von Dietrich Ayala und von der Firma NuSphere gesponsert. <http://dietrich.ganx4.com/nusoap/>

¹⁰ Urklasse der SOAP Implementierungen in PHP und Basis der meisten anderen SOAP Klassen.

```
$service = new Service();
$soap->addObjectMap($service, "urn:soapservice");
$soap->service($HTTP_RAW_POST_DATA);
```

Abbildung 8: Ein einfacher Web Service erstellt mit PEAR:SOAP

Abbildung 8 zeigt einen solchen Web Service. Die Klasse *Service* enthält eine Methode, die einen Parameter entgegen nimmt und diesen mit „Hello“ vorangestellt wieder zurückgibt.

Um diese Klasse nun mittels PEAR::SOAP als Web Service zur Verfügung zu stellen muss man zunächst die entsprechende Server-Klasse aus PEAR einbinden. Dies geschieht mit

```
require_once ("SOAP/Server.php");
```

Danach werden sowohl Server-Klasse als auch der Service selbst instanziiert:

```
$soap = new SOAP_Server();
```

```
$service = new Service();
```

Nun muss dem Server noch ein *Uniform Resource Name*¹¹ (URN) zugewiesen werden, dies geschieht mit folgendem Befehl:

```
$soap->addObjectMap($service, "urn:soapservice");
```

Der Aufruf `$soap->service($HTTP_RAW_POST_DATA)` bewirkt abschließend, dass der Service die Daten, die per POST übermittelt werden, abfängt und auswertet.

4 Vergleichbare Technologien

4.1 XML-RPC

XML-RPC stellt den Versuch dar die bekannten und etablierten Technologien XML und HTTP miteinander zu verbinden, um dadurch verteilte Anwendungen möglich zu machen. Das Prinzip ist relativ simpel, ein Rechner schickt einen Methodenaufruf verpackt in einer XML-Nachricht an einen anderen Rechner über HTTP und der andere Rechner antwortet wiederum ebenfalls mit einer XML-Nachricht. Es wird also eine entfernte Methode oder Prozedur aufgerufen und mittels XML kommuniziert, d.h. es findet ein *Remote Procedure Call* (RPC) statt. An dieser Stelle erkennt man schon die Verwandtschaft von XML-RPC und

¹¹ Eindeutiger Name einer Ressource

SOAP, die auf demselben Prinzip basieren. Genauer gesagt ist SOAP eine Weiterentwicklung von XML-RPC und RPC ist ein wichtiges Anwendungsgebiet von SOAP.

4.2 REST

Der *Representational State Transfer* (REST) steht für die Verwendung von HTTP ohne zusätzliche Nachrichtenprotokolle und deren Overhead. REST wurde von dem HTTP Mitentwickler Roy Thomas Fielding in seiner Dissertation beschrieben (vgl. [Fielding00], Kapitel 5, S. 76ff).

REST basiert auf der Tatsache, dass sich jede Ressource eindeutig mit einer *Uniform Resource Indicator*¹² (URI) identifizieren lässt. Der Zugriff auf die Ressource erfolgt dann mittels der vorhandenen Verben GET, PUT, POST und DELETE über HTTP.

Allerdings stellt REST noch keinen vollwertigen Ersatz für Web Services dar, da zum einen ein Verzeichnis zum Nachschlagen für REST basierte Dienste fehlt und zum anderen die HTTP Anweisungen PUT und DELETE kaum implementiert sind (vgl. [Wenz04], S. 29).

Dennoch gibt es im SOAP 1.2 Standard eine Annäherung an das REST-Prinzip, denn hier wird die Verwendung von SOAP über HTTP-GET beschrieben¹³ (vgl. [Wenz04], S. 29).

4.3 CORBA

Die *Common Object Request Broker Architecture* (CORBA) funktioniert als Mittelschicht (Middleware) zwischen Anwendungen, um so die Interaktion unterschiedlichster Software-Systeme gewährleisten zu können. Die Anwendungen benutzen genormte Schnittstellen zwischen den einzelnen Programmmodulen. Diese Schnittstellen werden durch die *Interface Definition Language* (IDL) beschrieben. Die CORBA Spezifikation wird von der *Object Management Group* (OMG) verwaltet.

Im Vergleich zu Web Services gilt CORBA allerdings als recht kompliziert.

¹² Überbegriff zu URL und URN

¹³ Eine detaillierte Gegenüberstellung von SOAP und REST kann unter http://www.prescod.net/rest/rest_vs_soap_overview/ gefunden werden.

4.4 Remoting

Unter Remoting werden hier Technologien verstanden, welche eine Kommunikation über Remote-Objekte erlauben, d.h. Objekte auf entfernten Rechnern können in derselben Weise angesprochen werden wie Objekte, die auf demselben Rechner laufen.

Bekannte Beispiele für diese Technologien sind Java RMI und .NET Remoting. Da die Objekte als Binärdaten übertragen werden ergibt sich gegenüber Web Services meistens ein Geschwindigkeitsvorteil. Der größte Nachteil dieser Technologien ist allerdings die Beschränkung auf die jeweilige Sprache, denn es können zum Beispiel nur Java Objekte über Java RMI miteinander kommunizieren.

5 Prototypische Umsetzung

5.1 Anforderungen

Die hier vorgestellten Web Service Technologien sollen nun bei der Entwicklung eines Prototyps zum Einsatz kommen. Dieser Prototyp soll zum einen als Client für andere Web Services dienen und zum anderen seine Funktionalität wiederum anderen Clients als Web Service zur Verfügung stellen (siehe Abbildung 9).

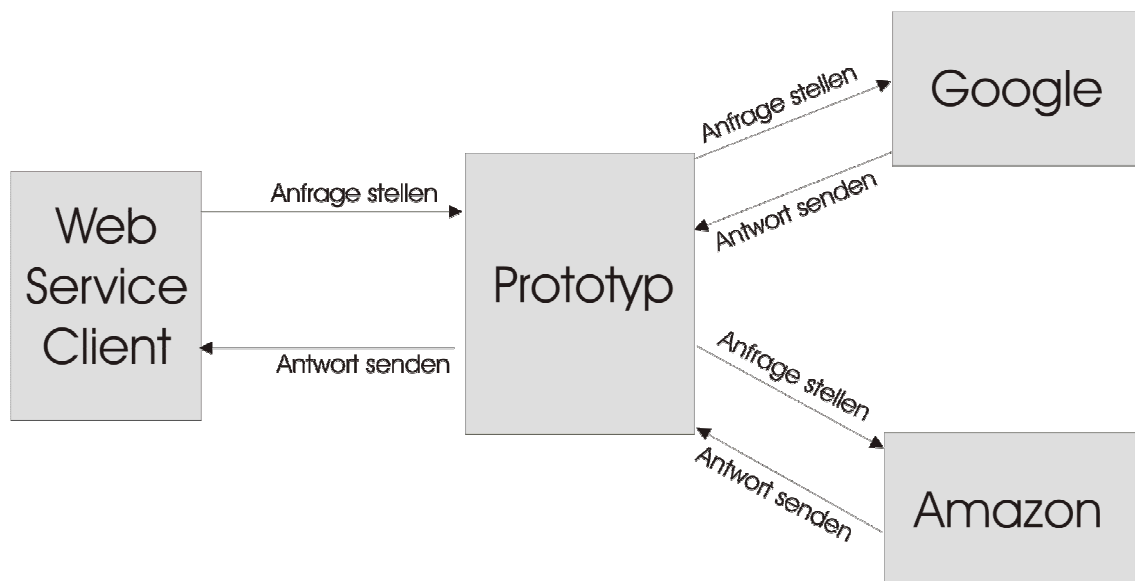


Abbildung 9: Verdeutlichung der Client/Server Funktionalität des Prototyps

Der Prototyp soll eine plattformübergreifende Suche nach Wissensressourcen ermöglichen. Dadurch sollen Personen auf kürzestem Wege Informationen zu einem bestimmten Wissens-

bereich, welcher zum Beispiel über einige Stichworte identifiziert wird, erhalten. Zunächst wurden die Dienste von Amazon und Google als Datenquellen eingebunden. Somit werden zu einem bestimmten Thema alle Webseiten und Artikel angezeigt, die bei Google und Amazon vorhanden sind.

In der heutigen Zeit hat man niemals „ausgelernt“, immer neue Anforderungen am Arbeitsplatz fordern eine ständige Weiterbildung der Mitarbeiter sowohl in Schulungen als auch in Eigeninitiative. Der hier vorgestellte Prototyp soll den Anwender durch den gleichzeitigen Zugriff auf verschiedene Dienste bei der Suche nach relevanten Lernmaterialien unterstützen und diese beschleunigen.

Die Qualität der Ergebnisse hängt natürlich zu einhundert Prozent von den benutzten Diensten ab, da der Prototyp deren Ergebnisse ja nur zusammenfasst und weiterleitet. Daher fiel die Wahl der benutzten Dienste auf Amazon und Google.

Google kann mit über vier Milliarden indizierten Seiten und seinem PageRank Algorithmus zur Bestimmung der Relevanz bestimmter Seiten für verschiedene Themen wohl als Marktführer unter den Suchmaschinen bezeichnet werden. Für die Qualität der Google Suche spricht ebenfalls, dass mehrere andere Suchmaschinen ebenfalls dieselbe Technik wie Google einsetzen.

Amazon gehört zu den größten Online-Buchhändlern der Welt und ist seit 1998 auch in Deutschland vertreten. Mittlerweile ist Amazon aber kein reiner Buchhändler mehr sondern vertreibt auch Elektronikartikel, Filme, Musik und diverse andere Artikel. Neben der Menge der angebotenen Waren sprechen auch die umfangreichen Suchfunktionen für den Einsatz der Amazon Web Services in diesem Prototyp.

5.2 Architektur

Der Prototyp wurde unter Benutzung der PEAR::SOAP API entwickelt. Zum Betrieb des Prototyps wird also sowohl die PEAR Bibliothek mit installiertem SOAP-Package als auch eine PHP Version größer als 4.3.0 benötigt.

Ziel der Architektur ist es eine Erweiterung des Prototyps um neue Dienste so einfach wie möglich zu gestalten. Es soll möglich sein beliebige weitere Clients zu implementieren und einzubinden, indem man sich einen der beiden bereits implementierten Clients zum Vorbild nimmt und diese in demselben Verzeichnis ablegt. Alle dort vorhandenen Klassen werden automatisch eingebunden und instanziiert.

Allgemeine Funktionen, wie zum Beispiel das Senden einer SOAP Anfrage, die Fehlerbehandlung und das Speichern der Antworten, werden von der Klasse *WS_Client* implementiert und von den Client Klassen *Amazon_Client* und *Google_Client* per Vererbung in Anspruch genommen (siehe Abbildung 10).

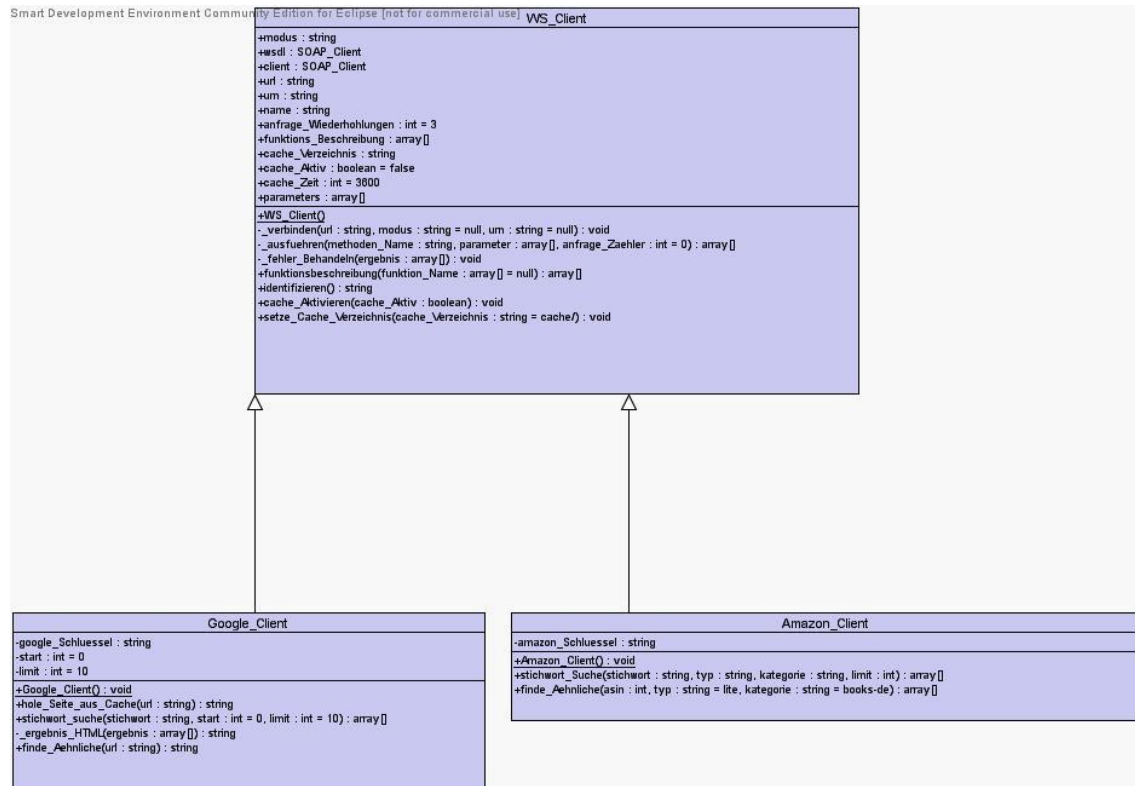


Abbildung 10: Klassendiagramm der Web Service Clients

5.3 Implementierungsdetails

Im Zuge der Implementierung des Prototyps deuteten erste Performanzschwächen darauf hin, dass der Einsatz in einem Produktivsystem mit mehreren tausend Anfragen pro Tag ohne weiteres nicht möglich ist. Die Beschränkungen der Google und Amazon Web Services mit 1000 Anfragen pro Tag, bzw. nur einer Anfrage pro Sekunde verhindern diesen Einsatz ebenfalls. Sollten in Zukunft weitere Dienste in den Prototyp integriert werden, würde sich dies ebenfalls negativ auf die Geschwindigkeit der Applikation auswirken. Um all diesen Aspekten entgegenzuwirken wurde ein einfacher Cache-Mechanismus implementiert. Dieser Mechanismus bewirkt, dass alle Antworten eines entfernten Methodenaufrufs, also die SOAP Response Nachrichten, zwischengespeichert werden. Sollte nun innerhalb eines frei definierbaren

Zeitraums dieselbe Methode mit denselben Parametern erneut aufgerufen werden, wird keine neue Anfrage gesendet sondern die alte Antwort aus dem Cache gelesen und ausgewertet.

5.4 Die Google API

Google ermöglicht den Zugriff auf bestimmte Funktionen der Suchmaschine per Web Service. Dazu muss man sich zunächst bei Google registrieren, um einen Schlüssel zu erhalten ohne den der Zugriff auf die Google Web Services überhaupt nicht möglich wäre. Dieser Schlüssel muss bei jedem Methodenaufruf zusätzlich zu den Parametern der Methode übermittelt werden, dadurch hat man die Möglichkeit bis zu eintausend Suchanfragen pro Tag an Google zu senden.

Im Einzelnen hat Google folgende Methoden für den entfernten Aufruf freigegeben:

- *doGoogleSearch*, die normale Suchanfrage
- *doGetCachedPage*, eine Seite aus dem Google-Cache abrufen
- *doSpellingSuggestion*, Vorschlag einer Korrigierung der Suchbegriffe

Abbildung 11 zeigt den Aufruf der entfernten Methode *doGoogleSearch* mit den entsprechenden Parametern. In diesem Beispiel wird nach dem Wort „Suchstring“ gesucht auf allen deutschen Internetseiten ('lr' => 'lang_de') gesucht.

```
$google = new
SOAP_Client("http://api.google.com/search/beta2");
$params = array ('key' => "00000000000000000000000000",
    'q' => "Suchstring", // Suchbegriff
    'start' => 0, //Starte ab Ergebn. Nr.0
    'maxResults' => 10, //gib max. 10 Ergebnisse zurück
    'filter' => TRUE, //ähnliche Ergebnisse verstecken
    'restrict' => '', // weitere Beschränkungen
    'safeSearch' => FALSE, //mit "Adult content"
    'lr' => 'lang_de', //nur deutsche Seiten
    'ie' => '', //input encoding -> deprecated
    'oe' => ''); //output enc. -> deprecated

$result = $google -> call("doGoogleSearch", $param,
'urn:GoogleSearch');
```

Abbildung 11: Eine einfache Suchanfrage an Google

Als Besonderheit sollte noch erwähnt werden, dass Google alle Daten mit UTF-8 kodiert hat. Dementsprechend muss der Wert des Parameters „q“ vor dem Aufruf der entfernten Methode *doGoogleSearch* mit dem Befehl `utf8-encode` kodiert werden. Analog dazu müssen auch die von Google übermittelten Ergebnisse mittels `utf8-decode` übersetzt werden, da ansonsten zum Beispiel die deutschen Umlaute nicht richtig übermittelt würden.

5.5 Die Amazon API

Die von Amazon über Web Services zur Verfügung gestellte Funktionalität übertrifft die von Google bei weitem. Man kann nicht nur den gesamten Amazon Produktkatalog nach vielen verschiedenen Kriterien durchsuchen, sondern es besteht ebenfalls die Möglichkeit Artikel in den Warenkorb zu übernehmen und eigene Artikel zum Verkauf anzubieten.

In dem hier vorgestellten Prototyp sollen vor allem die Suchfunktionen für Bücher zum Einsatz kommen. Daher wird hier auf die anderen Funktionen nicht weiter eingegangen. Eine vollständige Dokumentation der Amazon-API liegt dem Amazon Web Service Development Kit bei, welches unter <http://www.amazon.com/webservices> per download verfügbar ist. Unter dieser Adresse muss man sich genau wie bei Google registrieren um einen Schlüssel zu erhalten, welcher dann bei jedem Aufruf übermittelt werden muss. Als weitere Beschränkung gibt Amazon an, dass pro Sekunde nur eine Anfrage von einem Client akzeptiert wird.

Amazon bietet seine Web Services mit unterschiedlichen Kommunikationsprotokollen an. Man hat die Möglichkeit zwischen REST und SOAP wählen. Da bei Google bereits SOAP als Kommunikationsprotokoll zum Einsatz kommt, bot es sich an auch bei Amazon auf dieses Protokoll zu setzen.

```

$wsdl = new SOAP_WSDL("http://soap-
eu.amazon.com/schemas3/AmazonWebServices.wsdl");
$amazon = &$wsdl->getProxy();
$params = array ('devtag' => "000000000000000000",
    'mode' => "books-de",
    'locale' => "de",
    'type' => "lite",
    'keyword' => "Suchstring",
    'page' => "1" );
$result = $amazon->KeywordSearchRequest($params);

```

Abbildung 12: Eine Suche nach deutschen Büchern mit dem Stichwort „Suchstring“

Abbildung 12 zeigt eine einfache Suche nach allen deutschen Büchern im Amazon Produktkatalog mit einem bestimmten Stichwort. Für jede Anfrage werden maximal zehn Ergebnisse zurückgegeben. Sollten mehr vorhanden sein kann man durch das Erhöhen des *page*-Parameters auch die restlichen Ergebnisse abfragen.

Amazon bietet noch weitere Methoden zur Suche an. So kann man zum Beispiel mit der Methode *AuthorSearchRequest* Bücher auch nach Autoren durchsuchen oder mittels *BrowseNodeSearchRequest* nur die Artikel eines so genannten *Browse Nodes* anzeigen lassen. Amazon verwendet die *Browse Nodes* zur Kategorisierung der Artikel, somit sind zum Beispiel die Fachbücher unter dem *Browse Node* mit der ID 288100 zusammengefasst. Eine interessante Funktion bietet die Methode *SimilaritySearchRequest*, mit deren Hilfe man sich zu einem bestimmten Artikel alle inhaltlich verwandten Produkte anzeigen lassen kann.

6 Zusammenfassung und Ausblick

In diesem Dokument wurden die wichtigsten Standards im Umfeld der Web Service Technologie vorgestellt. Unter Verwendung dieser Standards wurde ein Prototyp entwickelt, der eine Unterstützung bei dem so genannten „lebenslangen Lernen“ bietet. Mit Hilfe von Web Services wurde ein plattformübergreifender Zugriff auf verschiedene Wissensressourcen ermöglicht. Im Zuge der Vorbereitung wurden verschiedene interessante Web Services auf deren Integrationsmöglichkeit in den Prototyp untersucht. Es wäre zum Beispiel auch interes-

sant nach bestimmten Büchern auf Ebay¹⁴, der weltweit größten Internet Auktionsplattform, zu suchen. Leider verhindert zum einen eine Lizenzgebühr, welche Ebay für den Zugriff auf ihr Produktivsystem verlangt und zum anderen die im Moment noch nicht vollständig vorhandene Kompatibilität von PEAR::SOAP zu den von Ebay verwendeten .NET Web Services diese Möglichkeit.

Eine weitere interessante Möglichkeit wäre eine Suche in der Citeseer-Datenbank¹⁵, welche unzählige wissenschaftliche Arbeiten verwaltet. Hier funktionierte die Registrierung¹⁶ für diesen Web Service leider zum Zeitpunkt der Erstellung dieser Arbeit nicht.

Sicher werden in Zukunft noch mehrere interessante Dienste ihre Ressourcen per Web Service zur Verfügung stellen. Aus diesem Grund wurde die Architektur des Prototyps so gewählt, dass eine Integration weiterer Dienste so einfach wie möglich gehalten wird.

Sicherlich gibt es bei der Funktionalität des Prototyps durchaus noch Verbesserungsmöglichkeiten. Es wäre zum Beispiel vorstellbar, Bücher direkt mit nur einem Mausklick bei Amazon zu bestellen.

Dieser Prototyp soll als einfach erweiterbarer Ansatz verstanden werden, welcher einen vereinheitlichten Zugriff auf vorhandene Wissensressourcen ermöglicht.

¹⁴ <http://www.ebay.de>

¹⁵ <http://citeseer.ist.psu.edu/>

¹⁶ http://smealsearch.psu.edu/api/soap_register.asp

- [Booth04] Booth, D., Haas, H., McCabe, F., Newcomer E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture, W3C Working Group Note 11. Februar 2004. <http://www.w3c.org/TR/ws-arch/>
- [Chappel02] Chappel, D., Jewell, T.: Java Web Services. O'Reilly, März 2002, 1. Auflage
- [Fielding00] Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, 2000. <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- [Gudgin03] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F.: SOAP Version 1.2 Part 2: Adjuncts, W3C Recommendation, 24 Juni 2003 <http://www.w3.org/TR/soap12-part2/>
- [Newcom02] Newcomer, E.: Understanding Web Services. Addison Wesley, 1. Mai 2002
- [Mitra03] Mitra, N.: SOAP Version 1.2 Part0: Primer, W3C Recommendation, 24. Juni 2003, <http://www.w3.org/TR/soap12-part0/>
- [Wenz04] Wenz, C., Hauser, T.: Web Services mit PHP. Galileo Press GmbH, Bonn 2004